Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Иметь возможность доступа из текущего объекта к любому объекту системы, «мечта» разработчика программы.

Расширить функциональность базового класса:

- метод переопределения головного объекта для текущего в дереве иерархии. Метод должен иметь один параметр, указатель на объект базового класса, содержащий указатель на новый головной объект. Переопределение головного объект для корневого объекта недопустимо. Недопустимо создать второй корневой объект. Недопустимо при переопределении, чтобы у нового головного появились два подчиненных объекта с одинаковым наименованием. Новый головной объект не должен принадлежать к объектам из ветки текущего. Если переопределение выполнено, метод возвращает значение «истина», иначе «ложь»;
- метод удаления подчиненного объекта по наименованию. Если объект не найден, то метод завершает работу. Один параметр строкового типа, содержит наименование удаляемого подчиненного объекта;
- метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задаться в следующем виде:
    - / - корневой объект;
    - //«имя объекта» - поиск объекта по уникальной имени от корневого (для однозначности уникальность требуется в рамках дерева);
    - . - текущий объект;
    - .«имя объекта» - поиск объекта по уникальной имени от текущего (для однозначности уникальность требуется в рамках ветви дерева от

текущего объекта);

- «имя объекта 1»[/«имя объекта 2»] . . . - относительная координата от текущего объекта, «имя объекта 1» подчиненный текущего;
- /«имя объекта 1»[/«имя объекта 2»] . . . - абсолютная координата от корневого объекта.

Примеры координат:

```
/
//ob_3
.
.ob_2
ob_2/ob_3
/ob_1/ob_2/ob_3
```

Если координата - пустая строка или объект не найден или определяется неоднозначно (дубляж имен на ветке, на дереве), тогда вернуть нулевой указатель.

Наименование объекта не содержит символы «.» и «/».

Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы. Единственное различие. В строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему. При построении дерева уникальность наименования относительно множества непосредственно подчиненных объектов для любого головного объекта необходимо соблюдать. Если это требование исходя из входных данных нарушается, то соответствующий подчиненный объект не создается.

Добавить проверку допустимости исходной сборки. Собрать дерево невозможно, если по заданной координате головной объект не найден (например, ошибка в наименовании или еще не расположен на дереве объектов). Если номер класса объекта задан некорректно, то объект не создается.

Собранная система отрабатывает следующие команды:

- SET «координата» – устанавливает текущий объект;
- FIND «координата» – находит объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» задает новый головной объект;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершает функционирование системы (выполнение программы).

Изначально, корневой объект для системы является текущим. При вводе данных в названии команд ошибок нет. Если при переопределении головного объекта нарушается уникальность наименований подчиненных объектов для нового головного, переопределение не производится.

1.1 Описание входных данных

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы. Единственное различие. В строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

После ввода состава дерева иерархии построчно вводятся команды:

- SET «координата» – установить текущий объект;
- FIND «координата» – найти объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» соответствует новому головному объекту;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершить функционирование системы (выполнение программы).

Команды SET, FIND, MOVE и DELETE вводятся произвольное число раз.

Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов:

```
rootela
/ object_1 3
/ object_2 2
/object_2 object_4 3
/object_2 object_5 4
/ object_3 3
/object_2 object_3 6
/object_1 object_7 5
/object_2/object_4 object_7 3
endtree
FIND object_2/object_4
SET /object_2
FIND //object_7
FIND object_4/object_7
FIND .
FIND .object_7
FIND object_4/object_7
MOVE .object_7
SET object_4/object_7
MOVE //object_1
MOVE /object_3
END
```

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева как в работе версия №2.

При ошибке определения головного объекта, прекратить сборку, вывести иерархию уже построенного фрагмента дерева, со следующей строки сообщение:

```
The head object «координата головного объекта» is not found
```

и прекратить работу программы с кодом возврата 1.

Если при построении при попытке создания объекта обнаружен дубляж, то вывести:

```
«координата головного объекта»     Dubbing the names of subordinate objects
```

Если дерево построено, то далее построчно вводятся команды.

Для команд SET если объект найден, то вывести:

```
Object is set: «имя объекта»
```

в противном случае:

```
The object was not found at the specified coordinate: «искомая координата объекта»
```

Для команд FIND вывести:

```
«искомая координата объекта»     Object name: «наименование объекта»
```

Если объект не найден, то:

```
«искомая координата объекта»     Object is not found
```

Для команд MOVE вывести:

```
New head object: «наименование нового головного объекта»
```

Если головной объект не найден, то:

```
«искомая координата объекта»     Head object is not found
```

Если переопределить головной объект не удалось, то:

```
«искомая координата объекта»     Redefining the head object failed
```

Если у нового головного объекта уже есть подчиненный с таким же именем, то вывести:

```
«искомая координата объекта»     Dubbing the names of subordinate objects
```

При попытке переподчинения головного объекта к объекту на ветке, вывести:

```
«координата нового головного объекта»      Redefining the head object failed
```

Для команды DELETE:

Если подчиненный объект удален, то вывести:

```
The object «абсолютный путь удаленного объекта» has been deleted
```

Если объект не найден, то ничего не выводить.

После команды END с новой строки вывести:

```
Current object hierarchy tree
```

Со следующей строки вывести текущую иерархию дерева.

Пример вывода иерархии дерева объектов:

```
Object tree
rootela
    object_1
        object_7
    object_2
        object_4
            object_7
        object_5
        object_3
    object_3
object_2/object_4     Object name: object_4
Object is set: object_2
//object_7     Object is not found
```

```
object_4/object_7     Object name: object_7
.     Object name: object_2
.object_7     Object name: object_7
object_4/object_7     Object name: object_7
.object_7     Redefining the head object failed
Object is set: object_7
//object_1     Dubbing the names of subordinate objects
New head object: object_3
Current object hierarchy tree
rootela
    object_1
        object_7
    object_2
        object_4
        object_5
        object_3
    object_3
        object_7
```

2 МЕТОД РЕШЕНИЯ

Для решения задача было добавлено/изменено:

Класс cl_base:

- Свойства/поля: нет изменений
- Методы:
  - cl_base* find_object_by_path (string) - для поиска объекта по его координате, модификатор доступа public
  - void remove_subordinate_object (string) - для удаления подчинённого объекта по его имени, модификатор доступа public
  - bool change_head (cl_base*) - для смены головного объекта, модификатор доступа public

Класс cl_application:

- Свойства/поля: изменений нет
- Методы: изменений нет

Класс cl_1:

- Свойства/поля: изменений нет
- Методы: изменений нет

Класс cl_2:

- Свойства/поля: изменений нет
- Методы: изменений нет

Класс cl_3:

- Свойства/поля: изменений нет
- Методы: изменений нет

Класс cl_4:

- Свойства/поля: изменений нет
- Методы: изменений нет

Класс cl_5:

- Свойства/поля: изменений нет
- Методы: изменений нет

Класс cl_6:

- Свойства/поля: изменений нет
- Методы: изменений нет

- Таблица 1 – Иерархия наследования классов

№	Имя класса	классы наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	cl_base			Базовый класс в дереве иерархии		
		cl_application	public			
		cl_1	public			
		cl_2	public			
		cl_3	public			
		cl_4	public			
		cl_5	public			

		cl_6	public			
2	cl_application			Класс приложения		
3	cl_1			Производный класс cl_base		
4	cl_2			Производный класс cl_base		
5	cl_3			Производный класс cl_base		
6	cl_4			Производный класс cl_base		
7	cl_5			Производный класс cl_base		
8	cl_6			Производный класс cl_base		

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.1 Алгоритм метода build_tree_objects класса cl_application

Функционал: Построение системы.

Параметры: нет .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "Object tree"	2
2		Объявление строковых переменных s_head_name, s_sub_name	3
3		Объявление переменной class_num типа int	4
4		Ввод s_head_name	5
5		Вызов метода set_name, в качестве параметра передаётся s_head_name	6
6		Ввод s_head_name	7
7		Инициализация указателя p_head на объект класса класса cl_base nullptr	8
8		Инициализация указателя last_created на объект класса cl_base адресом текущего объекта	9

№	Предикат	Действия	№ перехода
9	p_head_name!= "endtree"		10
			19
10		Ввод p_sub_name и class_num	11
11		Присваивание s_head возвращённого значения от метода find_object_by_path с параметром s_head_name	12
12	s_head found		13
			16
13	У объекта p_head нет подчинённого с именем s_sub_name		14
		Вывод перевода на следующую строку, data1 и "Dubbing the names of subordinate subjects"	18
14	class_num = 2	last_created := new cl_item2(p_head, data2)	18
	class_num = 3	last_created := new cl_item3(parent, data2)	18
	class_num = 4	last_created := new cl_item4(parent, data2)	18
	class_num =5	last_created := new cl_item5(parent, data2)	18
	class_num =6	last_created := new cl_item6(parent, data2)	18
			18
15		Вывод перевода на следующую строку, "The head object ", p_head_name и " is not found"	17
16		Вызов метода print_tree	15
17		Завершение программы с кодом возврата 1	∅

№	Предикат	Действия	№ перехода
18		Ввод p_head_name	19
19		Вызов метода print_tree	∅

3.2 Алгоритм метода exec_app класса cl_application

Функционал: Запуск системы.

Параметры: нет.

Возвращаемое значение: int - код возврата.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Обявление строковых переменных s_head_name, s_sub_name	2
2		Инициализация указателя current на объект класса cl_base адресом текущего объекта	3
3		Ввод s_head_name	4
4	s_head_name != "END"	Ввод s_sub_name	5
			24
5	s_head_name != "SET"	Инициализация указателя found на объект класса CL_BASE возвращённым от вызова метода FIND_OBJECT_BY_PATH с параметром s_sub_name у объекта	6

№	Предикат	Действия	№ перехода
		current адресом	
			8
6	found	current := found	7
			23
7		Вывод перевода на следующую строку, "The object was not found at the specified coordinate: ", s_sub_name	23
8	s_head_name != "FIND"	Вывод перевода на следующую строку, data2 и " "	9
			11
9		Инициализация указателя found на объект класса cl_base возвращённым от вызова метода find_object_by_path с параметром s_sub_name у объекта current адресом	10
10	found	Вывод "Object name: " и имени объекта found	23
		Вывод "Object is not found"	23
11	s_head_name != "MOVE"	Инициализация указателя new_head на объект класса cl_base возвращённым от вызова метода find_object_by_path с параметром s_sub_name у объекта current адресом	12
			15
1	new_head не найден	Вывод перевода на следующую строку, s_sub_name	23

№	Предикат	Действия	№ перехода
2		и " Head object is not found"	
			13
13	У объекта new_head есть подчинённый объект с именем объекта current	Вывод перевода на следующую строку, s_sub_name и " Dubbing the names of subordinate objects"	23
		Инициализация логической переменной res возвращённым от вызова метода change_head с параметром new_head у объекта current значением	14
14	res	Вывод перевода на следующую строку, "New head object: " и имени головного объекта current	23
		Вывод перевода на следующую строку, data2 и " Redefining the head object failed"	23
15	s_head_name = "DELETE"	Инициализация указателя obj_to_remove на объект класса cl_base возвращённым от вызова метода find_object_by_path с параметром s_sub_name у объекта current адресом	16
			23
16	obj_to_remove	Инициализация строковой переменной path_to_del значением ""	17
			23
17		Инициализация указателя head на объект класса cl_base значением obj_to_remove	18
18	У head есть головной объект	path_to_del := "/" + имя объекта head + path_to_del	19
			20

№	Предикат	Действия	№ перехода
19		Присвоение head возвращённого от вызова у head метода get_head	18
20	path_to_del = ""	path_to_del := "/"	21
0			21
21		Вывод перевода на следующую строку, "The object", path_to_del и " has been deleted"	22
22		Вызов метода remove_sub_object с параметром s_sub_name у объекта current	23
23		Ввод s_head_name	4
24		Вывод перевода на следующую строку и "The current object hierarchy tree"	25
25		Вызов метода print_tree	26
26		Возврат кода возврата успешного завершения	∅

3.3 Алгоритм функции main

Функционал: основная программа.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 4.

Таблица 4 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Инициализация объекта app класса cl_application созданным с использованием параметризованного конструктора объектом, в качестве параметра передаётся nullptr	2

№	Предикат	Действия	№ перехода
2		Вызов метода build_tree_objects у объекта app	3
3		Возврат возвращённого значения от метода exec_app у объекта app	∅

3.4 Алгоритм метода find_object_by_path класса cl_base

Функционал: Поиск объекта по его координате.

Параметры: string path.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода find_object_by_path класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация указателя root на объект класса cl_base адресом текущего объекта	2
2	У root есть головной объект	Присвоение root возвращённого от вызова у root метода get_p_head_object адреса	2
			3
3	Длина path = 0 или path = "."	Возврат адреса текущего объекта	∅
			4
4	path = "/"	Возврат root	∅
			5
5	в path найдено "//"	Инициализация строковой переменной obj_name	6

№	Предикат	Действия	№ перехода
		подстрокой path, взятой от 3 символа	
			7
6		Возврат адреса, возвращённого вызовом метода find_object_in_tree с параметром s_name	∅
7	в path найдено "."	Инициализация строковой переменной obj_name подстрокой path, взятой от 2 символа	8
			9
8		Возврат адреса, возвращённого вызовом метода find_object_in_branch с параметром s_name	∅
9	Первый символ path = '/'	Инициализация строковой переменной absolute_path подстрокой path, взятой от 1 символа	10
			11
10		Возврат возвращённого от вызова у root метода find_object_by_path с параметром absolute_path адреса	∅
11		Инициализация целочисленной переменной slash_index значением позиции символа '/' в строке path, либо -1	12
12	slash_index = -1	Возврат адреса подчинённого объекта с именем path, либо nullptr	∅
			13

№	Предикат	Действия	№ перехода
13		Инициализация строковой переменной subordinate_name подстрокой path, взятой до slash_index	14
14		Инициализация указателя sub на объект класса cl_base адресом подчинённого объекта с именем subordinate_name, либо nullptr	15
15	sub	Инициализация строковой переменной local_path подстрокой path, взятой от slash_index+1 символа	16
			17
16		Возврат адреса объекта, полученного вызовом метода find_object_by_path у объекта sub с параметром local_path	∅
17		Возврат nullptr	∅

3.5 Алгоритм метода remove_sub_object класса cl_base

Функционал: Удаление подчинённого объекта по имени.

Параметры: string .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода remove_sub_object класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация указателя obj_to_remove на объект класса cl_base полученным адресом от вызова метода get_subordinate_object_by_name с параметром name	2
2	obj_to_remove	Инициализация итератора it возвращённым значением от вызова функции find для поиска итератора удаляемого объекта	3
			∅
3		Удаление объекта obj_to_remove из subordinate_objects вызовом метода erase с параметром it	4
4		Демонтаж объекта obj_to_remove с использованием оператора delete	∅

3.6 Алгоритм метода change_head класса cl_base

Функционал: Смена головного объекта.

Параметры: cl_base* new_head.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода change_head класса cl_base

№	Предикат	Действия	№ перехода
1	new_head не задан	Возврат лжи	∅

№	Предикат	Действия	№ перехода
			2
2	Текущий объект является корневым	Возврат лжи	∅
			3
3	У объекта new_head есть подчинённый с именем значения s_object_name	Возврат лжи	∅
			4
4	У объекта new_head находится на ветке с текущим объектом	Возврат лжи	∅
			5
5		Инициализация итератора iterator возвращённым значением от вызова функции find для поиска текущего объекта в p_sub_objects объекта p_head_object	6
6		Удаление текущего объекта из subordinate_objects объекта p_head_object, используя метод erase с параметром iterator	7
7		p_head_object := new_head	8
8		Добавление текущего объекта в p_sub_objects объекта p_head_object	9

№	Предикат	Действия	№ перехода
9	p_head_object не готов	Вызов метода set_ready с параметром 0	10
			10
10		Возврат истины	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-8.

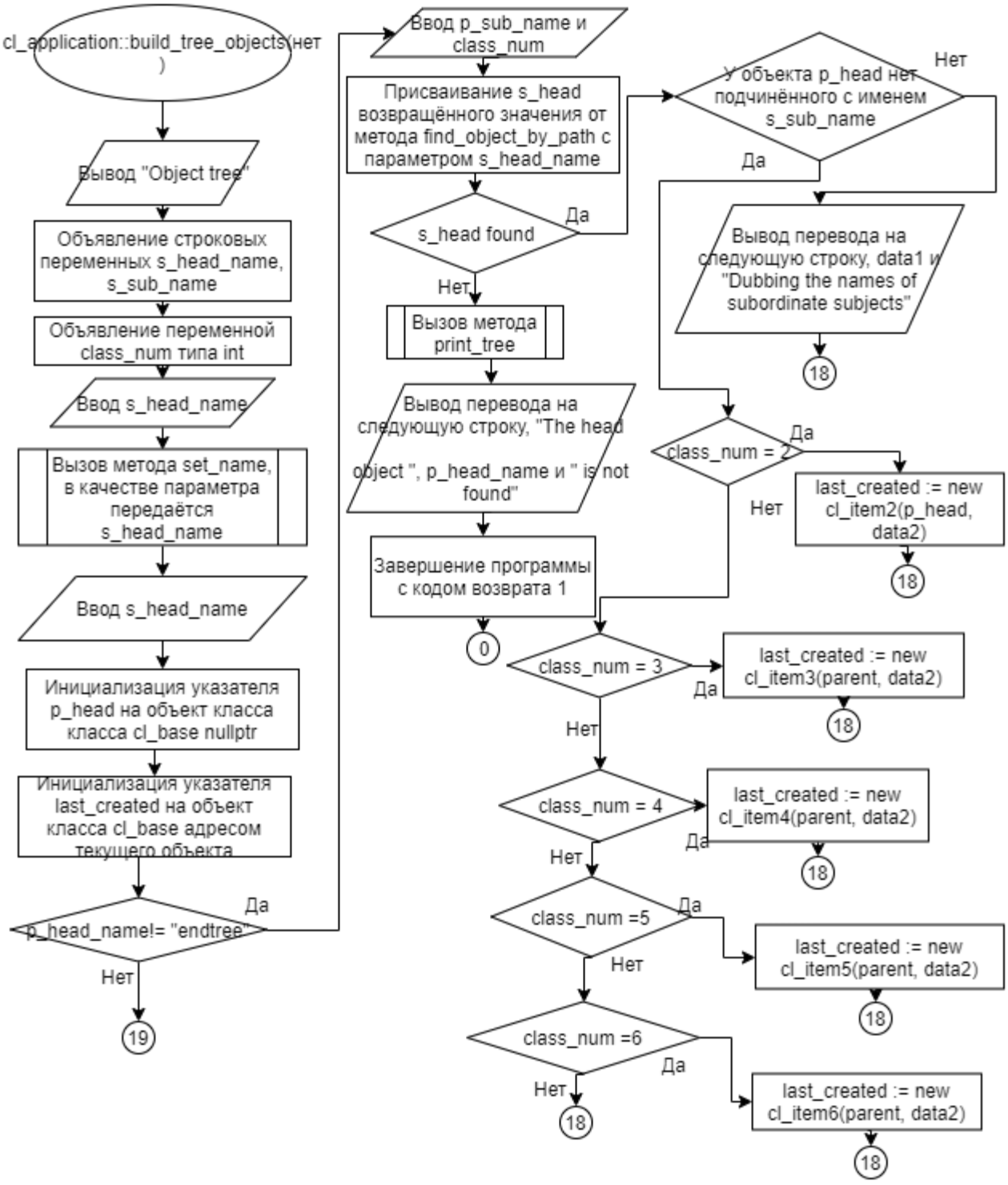

Рисунок 1 – Блок-схема алгоритма

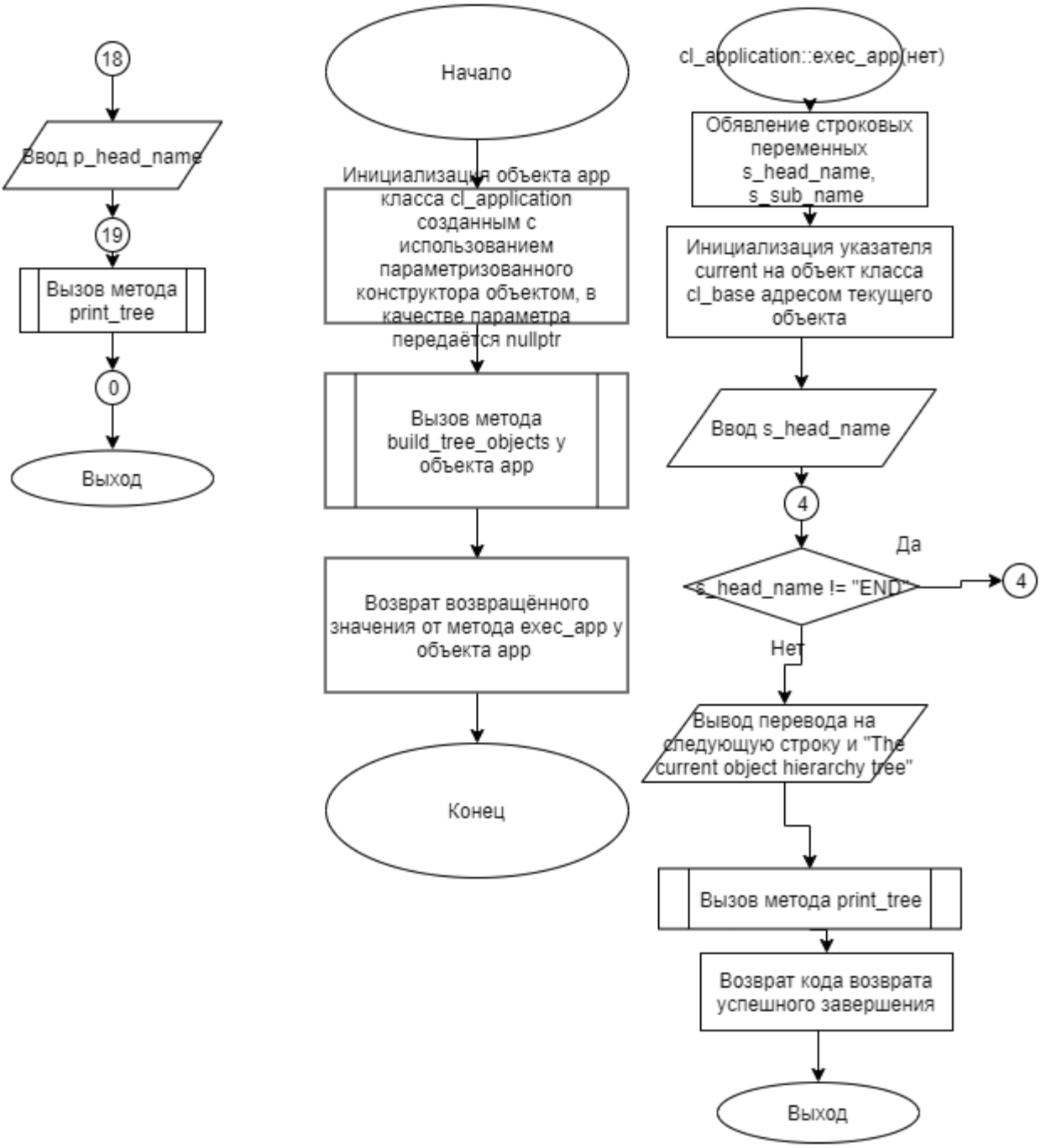

Рисунок 2 – Блок-схема алгоритма

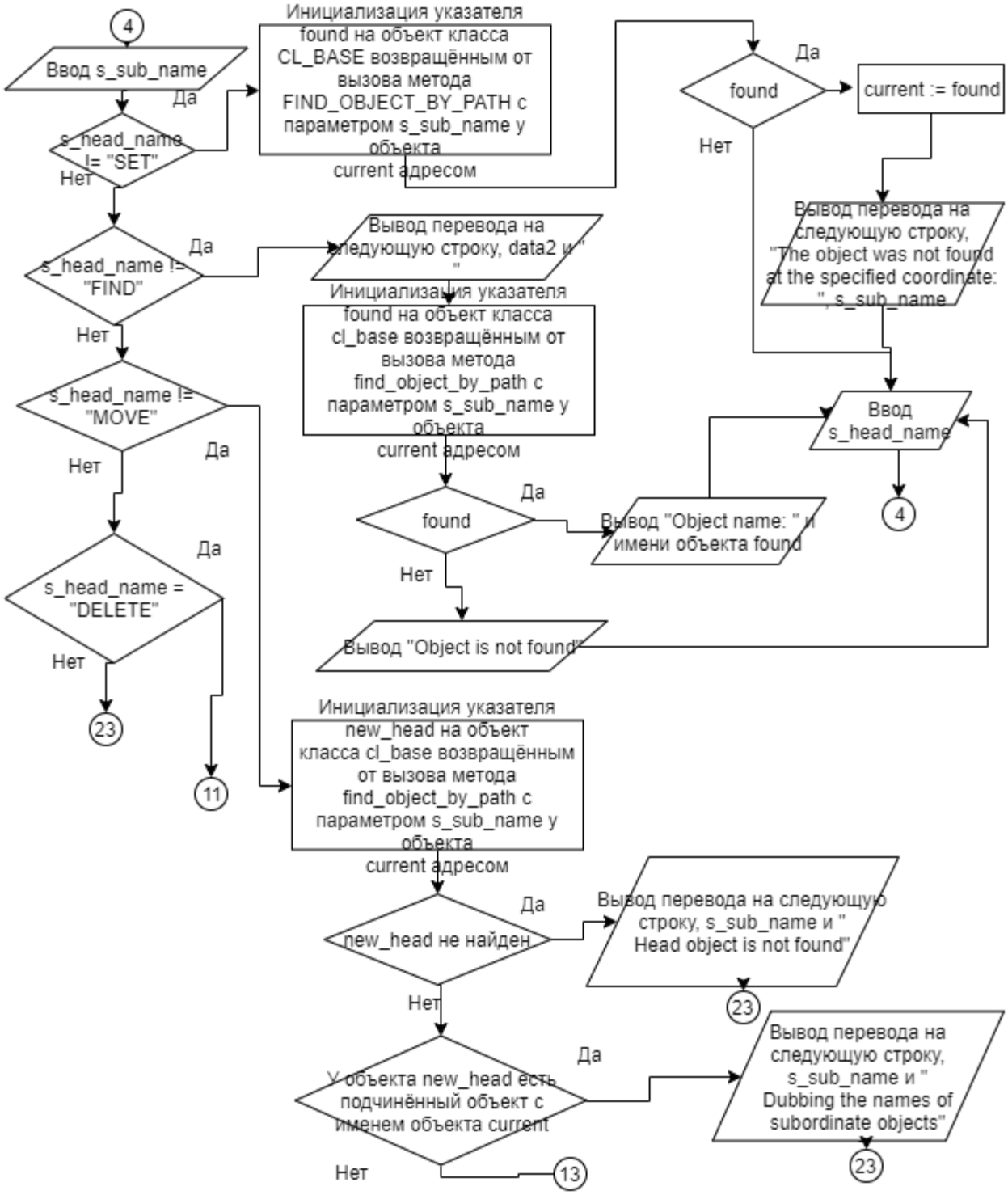

Рисунок 3 – Блок-схема алгоритма

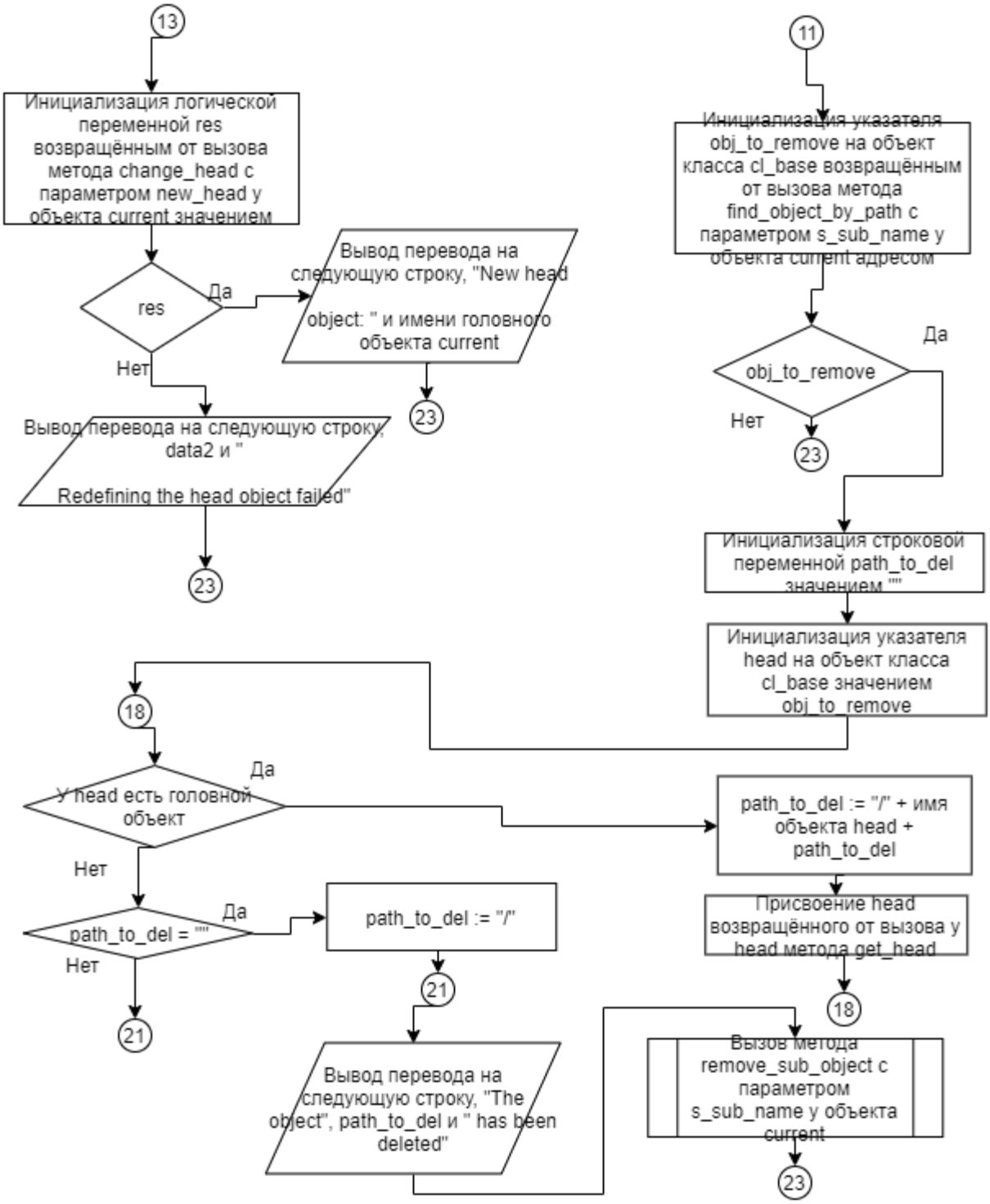

Рисунок 4 – Блок-схема алгоритма

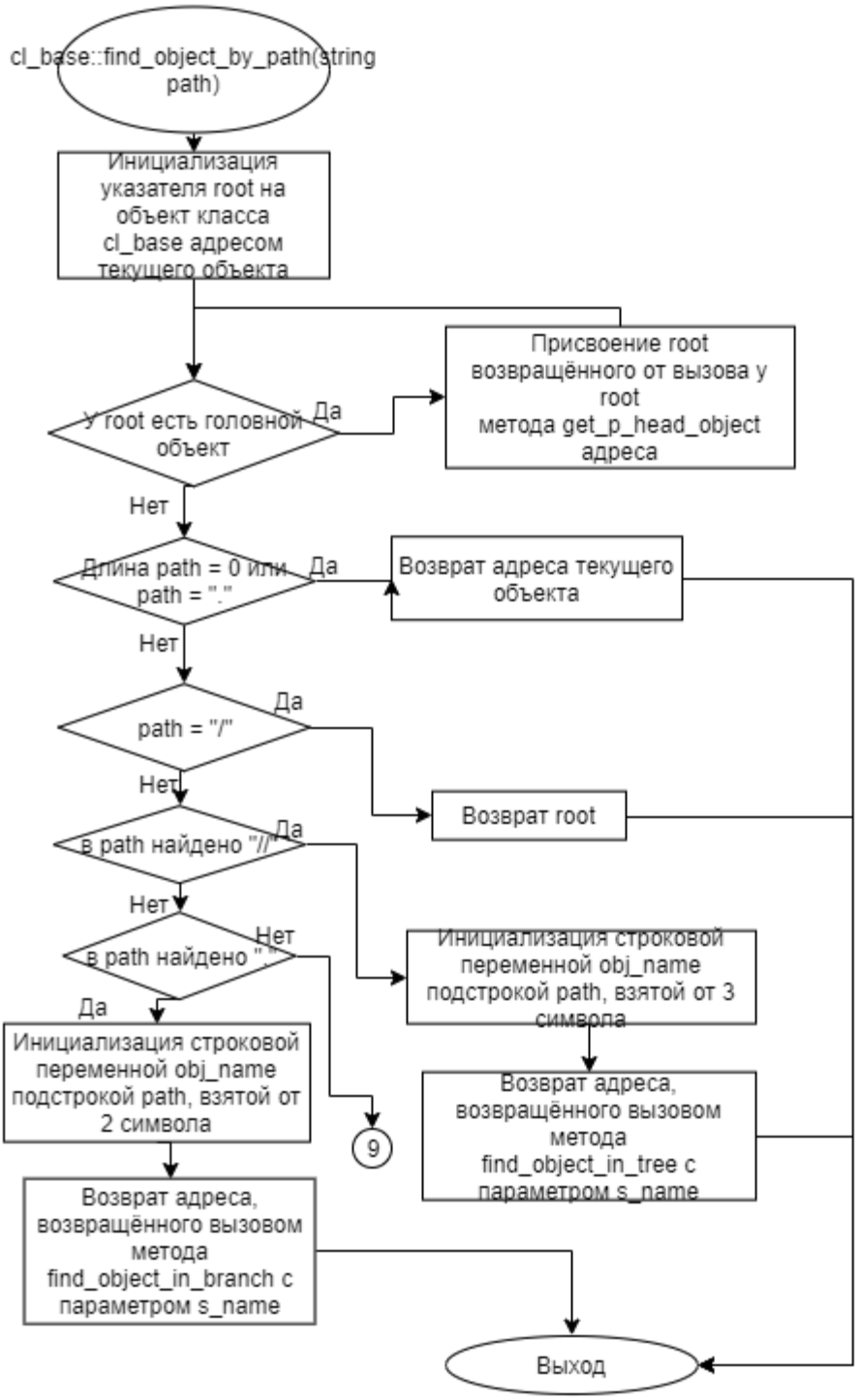

Рисунок 5 – Блок-схема алгоритма

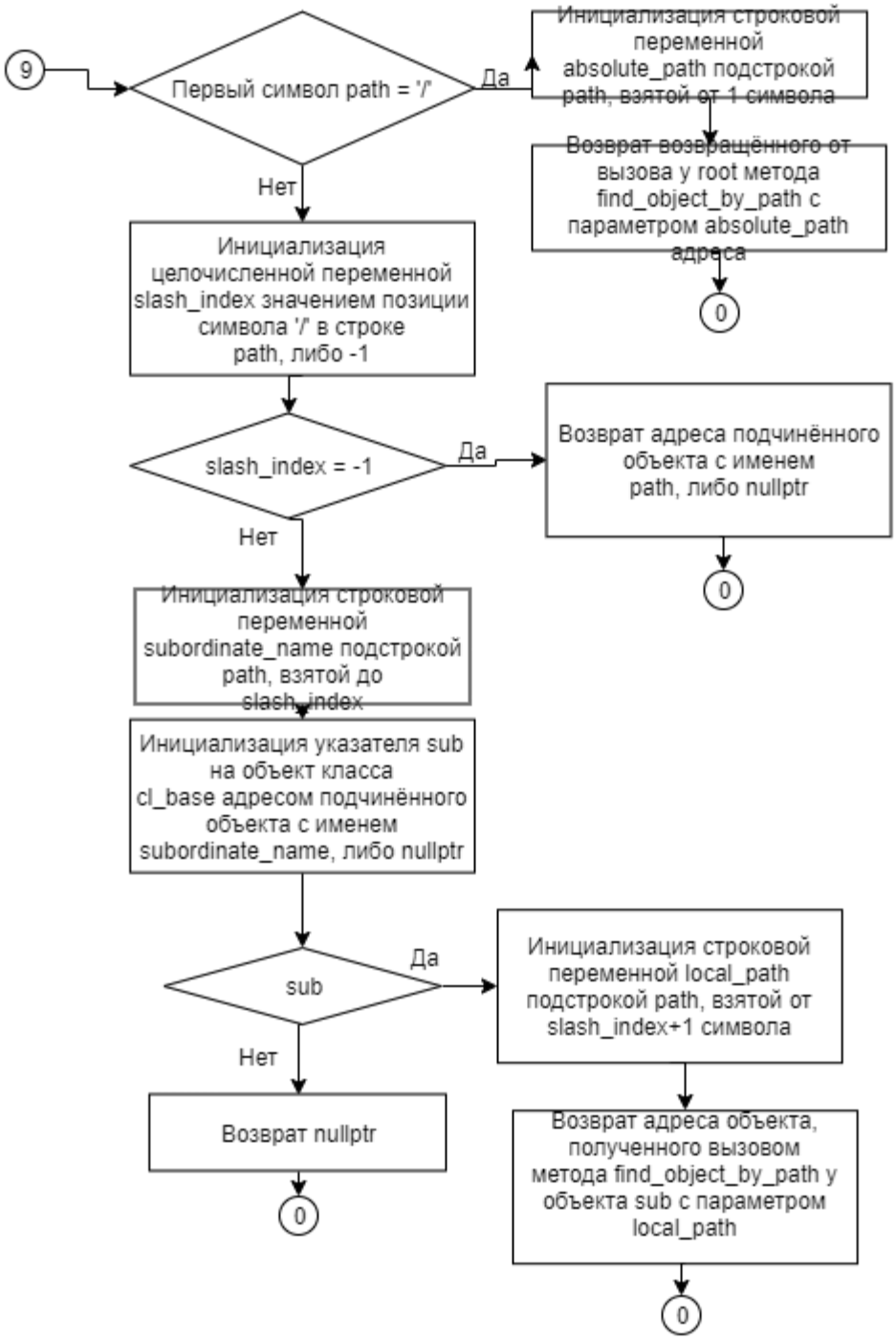

Рисунок 6 – Блок-схема алгоритма

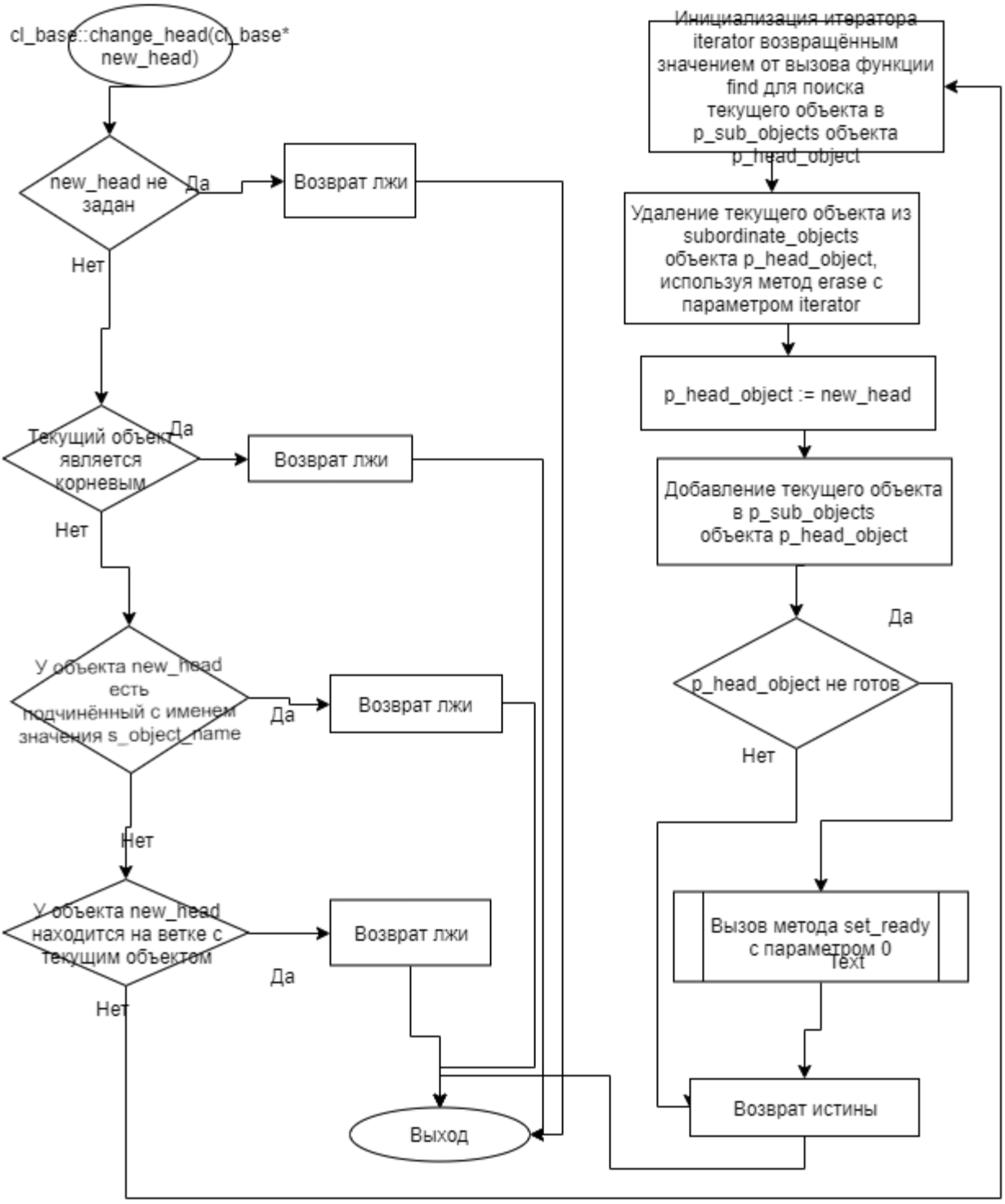

Рисунок 7 – Блок-схема алгоритма

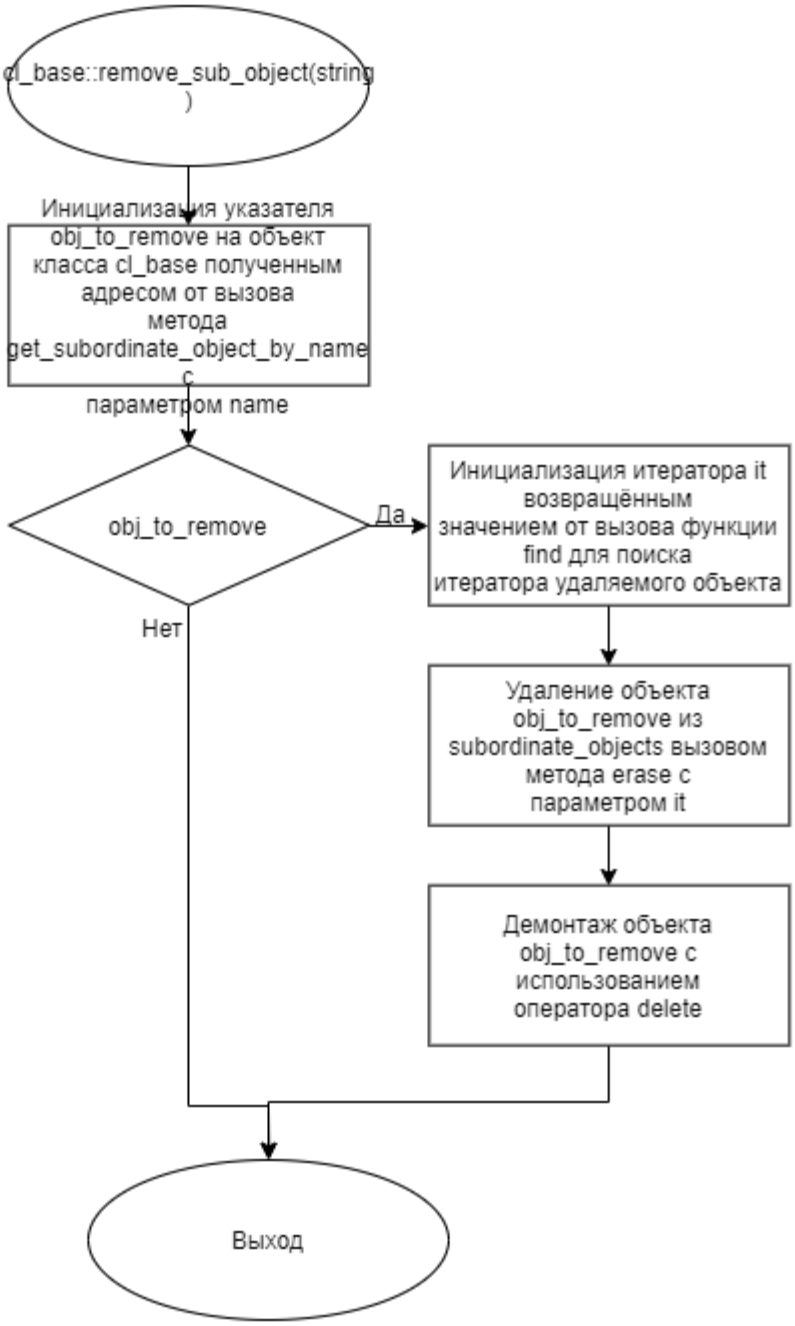

Рисунок 8 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"

cl_1::cl_1(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.2 Файл cl_1.h

Листинг 2 – cl_1.h

```
#ifndef CL_1__H
#define CL_1__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_1: public cl_base{
      public:
            cl_1(cl_base* p_head, string s_name);
};
#endif
```

5.3 Файл cl_2.cpp

Листинг 3 – cl_2.cpp

```
#include "cl_2.h"
#include <iostream>
using namespace std;

cl_2::cl_2(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.4 Файл cl_2.h

Листинг 4 – cl_2.h

```
#ifndef CL_2__H
#define CL_2__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_2: public cl_base{
      public:
            cl_2(cl_base* p_head, string s_name);
};
#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.6 Файл cl_3.h

Листинг 6 – cl_3.h

```
#ifndef CL_3__H
#define CL_3__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_3: public cl_base{
      public:
            cl_3(cl_base* p_head, string s_name);
};
#endif
```

5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.8 Файл cl_4.h

Листинг 8 – cl_4.h

```
#ifndef CL_4__H
#define CL_4__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_4: public cl_base{
      public:
            cl_4(cl_base* p_head, string s_name);
};


#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.10 Файл cl_5.h

Листинг 10 – cl_5.h

```
#ifndef CL_5__H
#define CL_5__H
#include "cl_base.h"
#include <iostream>
using namespace std;
```

```
class cl_5: public cl_base{
      public:
            cl_5(cl_base* p_head, string s_name);
};


#endif
```

5.11 Файл cl_6.cpp

Листинг 11 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base* p_head, string s_name):cl_base(p_head, s_name){}
```

5.12 Файл cl_6.h

Листинг 12 – cl_6.h

```
#ifndef CL_6__H
#define CL_6__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_6: public cl_base{
      public:
            cl_6(cl_base* p_head, string s_name);
};


#endif
```

5.13 Файл cl_application.cpp

Листинг 13 – cl_application.cpp

```
#include "cl_application.h"
#include <iostream>
using namespace std;

cl_application::cl_application(cl_base*     p_head_object,     string     s_name):
cl_base(p_head_object, s_name){}
```

```
void cl_application::build_tree_objects(){

	std::cout << "Object tree";
	std::string data1, data2;
	int data3;
	std::cin >> data1;
	this->set_name(data1);
	std::cin >> data1;
	cl_base* parent = nullptr;
	cl_base* last_created = this;
	while (data1 != "endtree")
	{
		std::cin >> data2 >> data3;
		parent = find_object_by_path(data1);
		if (parent)
		{
			if(!parent->get_sub_object(data2))
			{
				switch (data3)
				{
					case 2:
						last_created = new cl_1(parent, data2);
						break;
					case 3:
						last_created = new cl_2(parent, data2);
						break;
					case 4:
						last_created = new cl_3(parent, data2);
						break;
					case 5:
						last_created = new cl_4(parent, data2);
						break;
					case 6:
						last_created = new cl_5(parent, data2);
						break;
					default:
						break;
				}
			}
			else
			{
				std::cout << std::endl << data1 << "     Dubbing the names
of subordinate objects";
			}
		}
		else
		{
			print_tree();
			std::cout << std::endl << "The head object " << data1 << " is
not found";
			exit(1);
		}
		std::cin >> data1;
	}
	print_tree();
```

```
}


int cl_application::exec_app(){


      cl_base* current = this;
      std::string data1, data2;
      std::cin >> data1;
      while (data1 != "END")
      {
            std::cin >> data2;
            if (data1 == "SET")
            {
                  cl_base* found = current->find_object_by_path(data2);
                  if (found)
                  {
                        current = found;
                        std::cout << std::endl << "Object is set: " << current-
>get_name();
                  }
                  else
                  {
                        std::cout << std::endl << "The object was not found at the
specified coordinate: " << data2;
                  }
            }
            else if (data1 == "FIND")
            {
                  std::cout << std::endl << data2 << "     ";
            cl_base* found = current->find_object_by_path(data2);
                  if (found)
                  {
                        std::cout << "Object name: " << found->get_name();
                  }
                  else
                  {
                        std::cout << "Object is not found";
                  }
            }
            else if (data1 == "MOVE")
            {
                  cl_base* new_head = current->find_object_by_path(data2);
                  if (!new_head)
                  {
                        std::cout << std::endl << data2 << "      Head object is
not found";
                  }
                  else if (new_head->get_sub_object(current->get_name()))
                  {
                        std::cout << std::endl << data2 << "     Dubbing the names
of subordinate objects";
                  }
                  else
                  {
                        bool res = current->change_head(new_head);
```

```
                        if (res)
                        {
                              std::cout << std::endl << "New head object: " <<
                              current->get_head_object()->get_name();
                        }
                        else
                        {
                              std::cout << std::endl << data2 << "     Redefining
the head object failed";
                        }
                  }
            }
            else if (data1 == "DELETE")
            {
                  cl_base* obj_to_remove = current->get_sub_object(data2);
                  if(obj_to_remove)
                  {
                        std::string path_to_del = "";
                  cl_base* head = obj_to_remove;
                        while(head->get_head_object())
                        {
                              path_to_del = "/" + head->get_name() +path_to_del;
                              head = head->get_head_object();
                        }
                        if(path_to_del == "") path_to_del = "/";
                  std::cout << std::endl << "The object " << path_to_del <<
                  " has been deleted";
                  current->remove_sub_object(data2);
                  }
            }
            std::cin >> data1;
      }
      std::cout << std::endl << "Current object hierarchy tree";
      print_tree();
      return 0;


}
```

5.14 Файл cl_application.h

Листинг 14 – cl_application.h

```
#ifndef CL_APPLICATION__H
#define CL_APPLICATOIN__H
#include <iostream>
#include "cl_base.h"
#include "cl_1.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
```

```
using namespace std;

class cl_application: public cl_base{
	public:
		cl_application(cl_base* p_head_object, string s_name = "Base_Object");
		void build_tree_objects();
		int exec_app();
};
#endif
```

5.15 Файл cl_base.cpp

Листинг 15 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base* p_head_object, string s_name){
	this->p_head_object = p_head_object;
	this->s_name = s_name;
	this->state = 0;
	if(p_head_object) {
		p_head_object -> p_sub_objects.push_back(this); // добавление в состав
подчиненных головного объекта
	}
}

bool cl_base::set_name(string s_new_name){


	if (this->p_head_object && this-> p_head_object->get_sub_object(s_new_name))
	{
		return false;
	}

	this->s_name = s_new_name;
	return true;
}

string cl_base::get_name(){
	return this->s_name;
}

cl_base* cl_base::get_head_object(){
	return this->p_head_object;
}

void cl_base:: print_tree(int spaces){
	cout <<endl;
	for(int i = 0; i < spaces; i++)cout << "    ";
	cout<< s_name;
	for(int i = 0 ; i <p_sub_objects.size();i++){
		p_sub_objects[i]->print_tree(spaces +1);
	}
```

```
}

cl_base::~cl_base(){
	for (int i=0 ; i < this->p_sub_objects.size(); i++) {
			delete p_sub_objects[i];
		}
}

cl_base* cl_base::get_sub_object(string S_name){
	for (auto sub: p_sub_objects) {
			if (sub->get_name() == S_name) {
				return sub;
			}
		}
	return nullptr;
}



cl_base* cl_base::search_branch(string name, int* countptr){
	cl_base* found = nullptr;
	int count = 0;
	if (s_name == name)
	{
		found = this;
		count = 1;
	}
	int new_count = 0;
	if (!countptr)
	{
		countptr = &new_count;
	}
	for (auto subordinate_object : p_sub_objects)
	{
		cl_base* obj = subordinate_object->search_branch(name,countptr);
		if (*countptr)
		{
			return nullptr;
		}
		if (obj)
		{
			if (count++)
			{
				(*countptr)++;
				return nullptr;
			}
			found = obj;
		}
	}
	return found;

}

cl_base* cl_base::search_tree(string s_name){
```

```
	if (p_head_object) return p_head_object->search_tree(s_name);
	return search_branch(s_name);
}


void cl_base::print_status(int spaces){
	cout << endl;
	for (int i =0; i < spaces; i++)cout<< "    ";
	cout << s_name << (state ? " is ready" : " is not ready");
	for(auto sub: p_sub_objects){
			sub->print_status(spaces+1);
		}
}



void cl_base::set_status(int state){

	if (!state)
	{
		for (auto sub: p_sub_objects)
		{
		sub->set_status(0);
		}
		this->state= state;
	}
	else if (!p_head_object || p_head_object->state)
	{
		this->state = state;
	}

}




void cl_base::remove_sub_object(string name){

	cl_base* obj_to_remove = get_sub_object(name);
	if (obj_to_remove)
	{
		auto	it	=	std::find(p_sub_objects.begin(),p_sub_objects.end(),
obj_to_remove);
		p_sub_objects.erase(it);
		delete obj_to_remove;
	}


}


bool cl_base::change_head(cl_base* new_head){
	if (!new_head) return false;
	if (!p_head_object) return false;
	if (new_head->get_sub_object(s_name)) return false;
	cl_base* test_head = new_head;
```

```
	while (test_head)
	{
		if (test_head == this) return false;
		test_head = test_head->get_head_object();
	}
	auto  iterator  =  find(p_head_object->p_sub_objects.begin(),  p_head_object-
>p_sub_objects.end(), this);
	p_head_object->p_sub_objects.erase(iterator);
	p_head_object = new_head;
	p_head_object->p_sub_objects.push_back(this);
	if (!p_head_object->state)
	{
		set_status(0);
	}
	return true;
}

cl_base* cl_base::find_object_by_path(string path){
	cl_base* root = this;
	while (root->get_head_object())
	{
		root = root->get_head_object();
	}
	if (path == "/") return root;
	if(path.length() == 0 || path == ".") return this;
	if (path.find("//") != -1)
	{
		string obj_name = path.substr(2);
		return search_tree(obj_name);
	}
	if (path.find(".") != -1){
		string obj_name = path.substr(1);
		return search_branch(obj_name);
	}
	if (path[0] == '/'){
		string absolute_path = path.substr(1);
		return root-> find_object_by_path(absolute_path);
	}
	int slash_index = path.find('/');
	if (slash_index == -1)
	{
		return get_sub_object(path);
	}
	string subordinate_name = path.substr(0, slash_index);
	cl_base* sub = get_sub_object(subordinate_name);
	if (sub){
		string local_path = path.substr(slash_index + 1);
		return sub->find_object_by_path(local_path);
	}
	return nullptr;


}
```

5.16 Файл cl_base.h

Листинг 16 – cl_base.h

```
#ifndef CL_BASE__H
#define CL_BASE__H
#include <vector>
#include <string>
#include <algorithm>
#include <iostream>
#include <queue>
using namespace std;

class cl_base{

      string s_name;
      cl_base* p_head_object;
      vector <cl_base*> p_sub_objects;
      int state;

      public:
      cl_base(cl_base* p_head_object, string s_name= "Base_object");
      bool set_name(string s_new_name);
      string get_name();
      cl_base* get_head_object();
      void print_tree(int spaces=0);
      ~cl_base();
      cl_base* get_sub_object(string S_name);

      cl_base* search_branch(string s_name, int* countptr= nullptr);
      cl_base* search_tree(string s_name);
      void print_status(int spaces=0);
      void set_status(int state);


      cl_base* find_object_by_path(string path);
      void remove_sub_object(string name);
      bool change_head(cl_base* new_head);

};
#endif
```

5.17 Файл main.cpp

Листинг 17 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "cl_application.h"
using namespace std;
```

```
int main()
{
      cl_application ob_cl_application(nullptr);
      ob_cl_application.build_tree_objects();
      return ob_cl_application.exec_app();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 8.

Таблица 8 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

rootela
/ object_1 3
/ object_2 2
/object_2 object_4 3
/object_2 object_5 4
/ object_3 3
/object_2 object_3 6
/object_1 object_7 5
/object_2/object_4
object_7 3
endtree
FIND object_2/object_4
SET /object_2
FIND //object_7
FIND object_4/object_7
FIND .
FIND .object_7
FIND object_4/object_7
MOVE .object_7
SET object_4/object_7
MOVE //object_1
MOVE /object_3
END
	Object tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_7
	object_5
	object_3
	object_3
	object_2/object_4
	Object name: object_4
	Object is set: object_2
	//object_7 Object is not found
	object_4/object_7
	Object name: object_7
	. Object name: object_2
	.object_7 Object name: object_7
	object_4/object_7
	Object name: object_7
	.object_7 Redefining the head object failed
	Object is set: object_7
	//object_1 Dubbing the names of subordinate objects
	New head object: object_3
	Current object hierarchy tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_5
	object_3
	object_3
	object_7
		Object tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_7
		object_5
		object_3
		object_3
		object_2/object_4
		Object name: object_4
		Object is set: object_2
		//object_7 Object is not found
		object_4/object_7
		Object name: object_7
		. Object name: object_2
		.object_7 Object name: object_7
		object_4/object_7
		Object name: object_7
		.object_7 Redefining the head object failed
		Object is set: object_7
		//object_1 Dubbing the names of subordinate objects
		New head object: object_3
		Current object hierarchy tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_5
		object_3
		object_3
		object_7

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).